GQA

Grouped-Query Attention

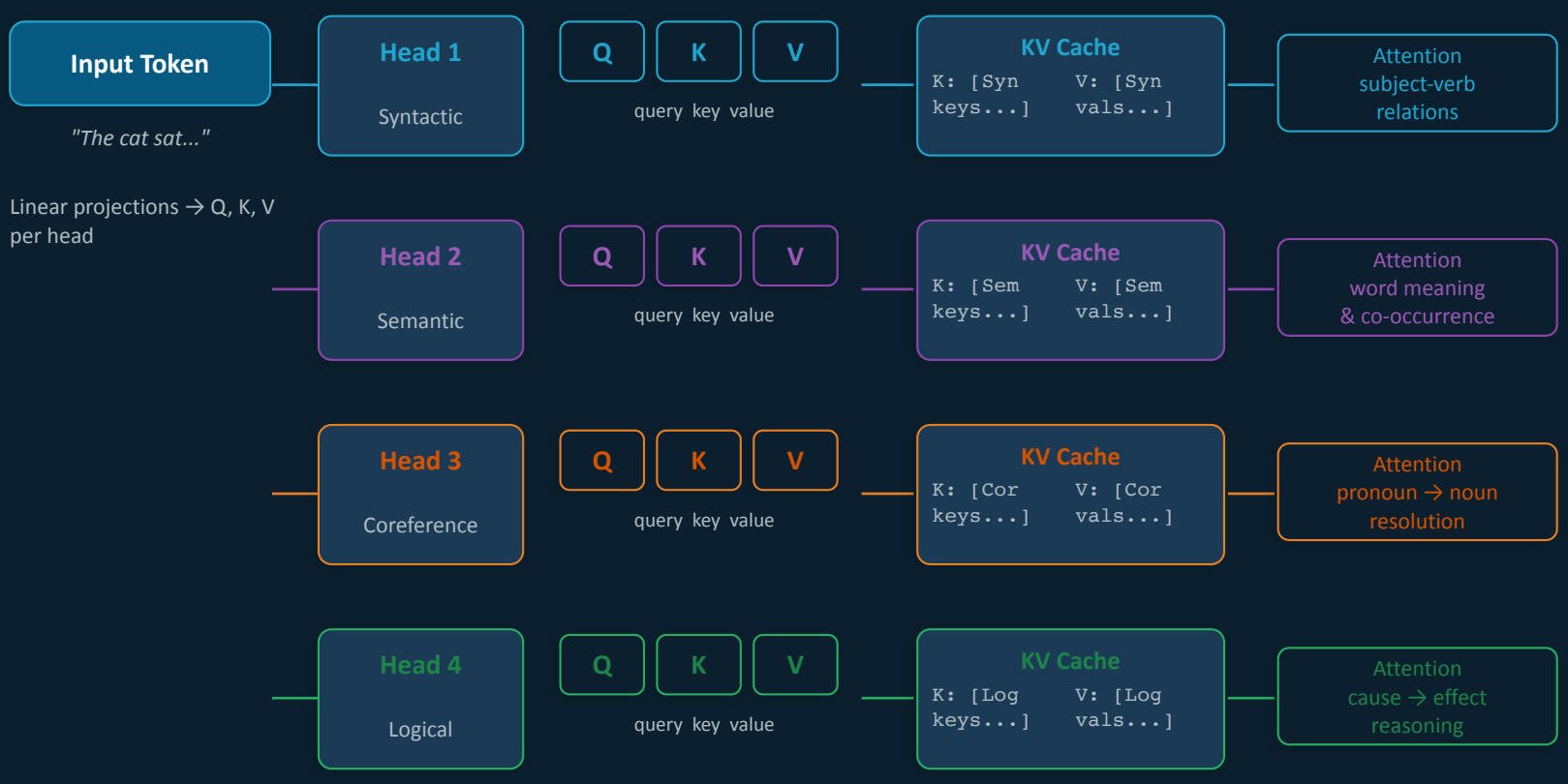
Training Generalized Multi-Query Transformer Models
from Multi-Head Checkpoints

Ainslie, Lee-Thorp, de Jong et al. — Google Research (2023)

Presented by: Sanskriti Goyal

What is Multi-Head Attention?

Each head independently attends to a different aspect of the input — syntax, semantics, coreference, logic

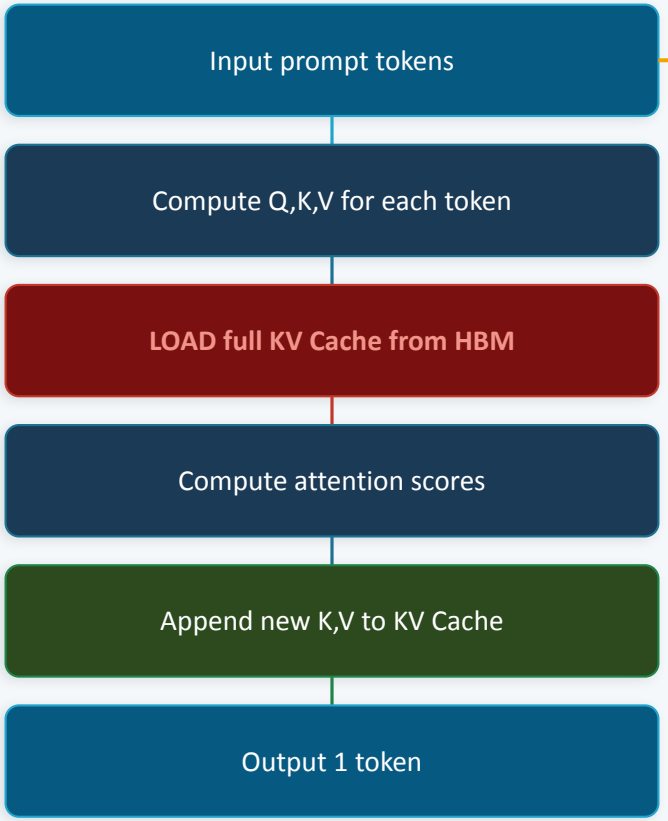


All 4 head outputs concatenated → final output

MHA: H heads × independent KV Cache = H × d_head × L × 2 values stored per layer

Why is MHA Inference Slow?

The autoregressive loop forces sequential token generation — and every step re-loads the entire KV Cache



KV Cache Size: T5-XXL

Size = 2 × H × d_head × L × batch

H (heads)	=	64	*T5-XXL attention heads*
d_head (dim)	=	128	*dimension per head*
L (seq len)	=	4,096	*context tokens*
batch	=	1	*single request*
dtype	=	FP16	*2 bytes per value*

≈ 2 GB loaded per decode step

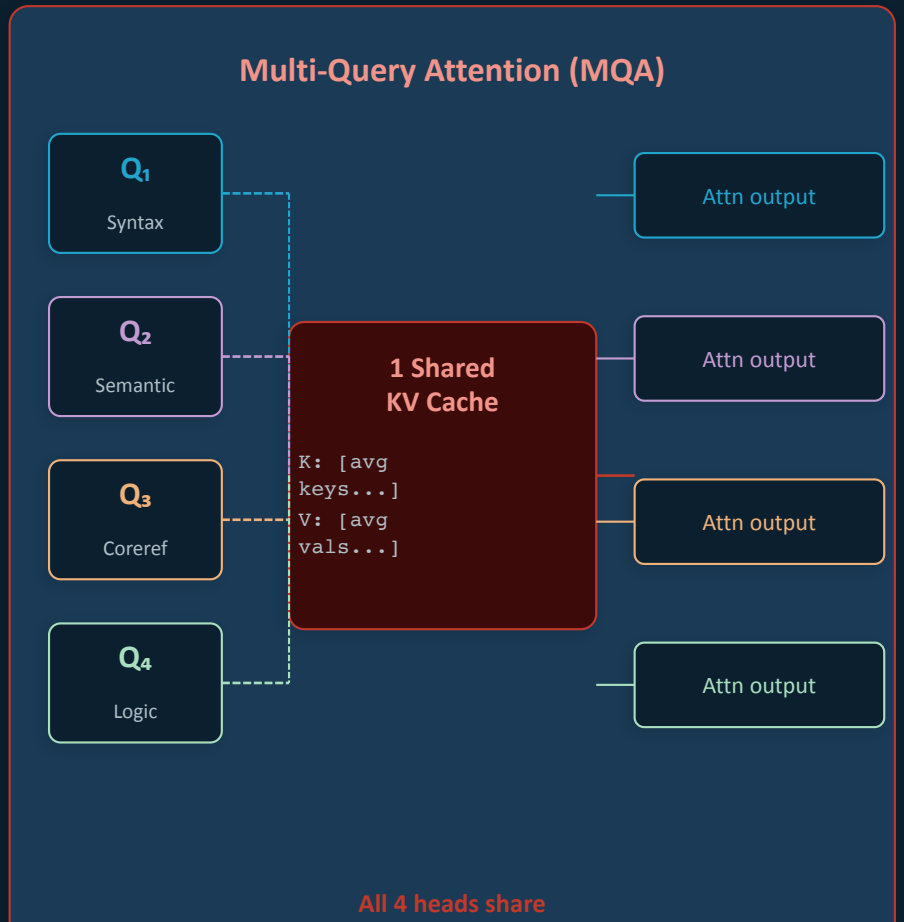
GPU peak bandwidth: 2 TB/s → ~1 ms just loading KV per token

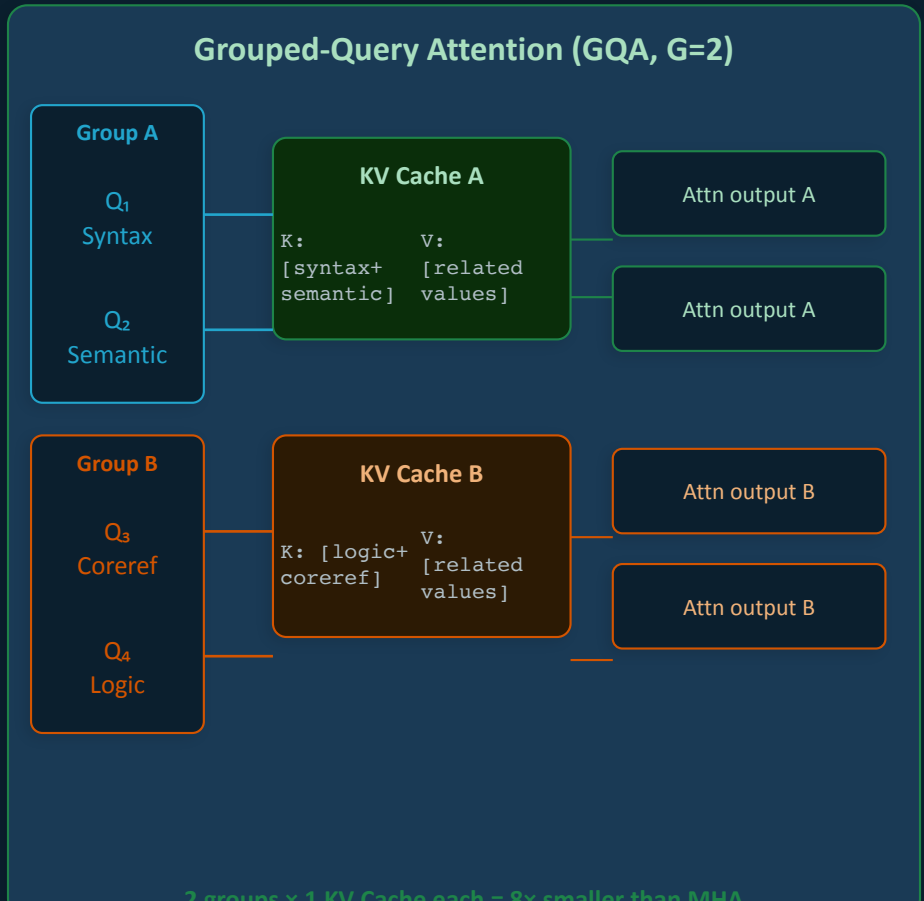
KV Cache = stored K,V tensors for all past tokens (the data structure)

KV Caching = the process of storing & reusing them across steps (the technique)

MQA and GQA: Detailed Architecture

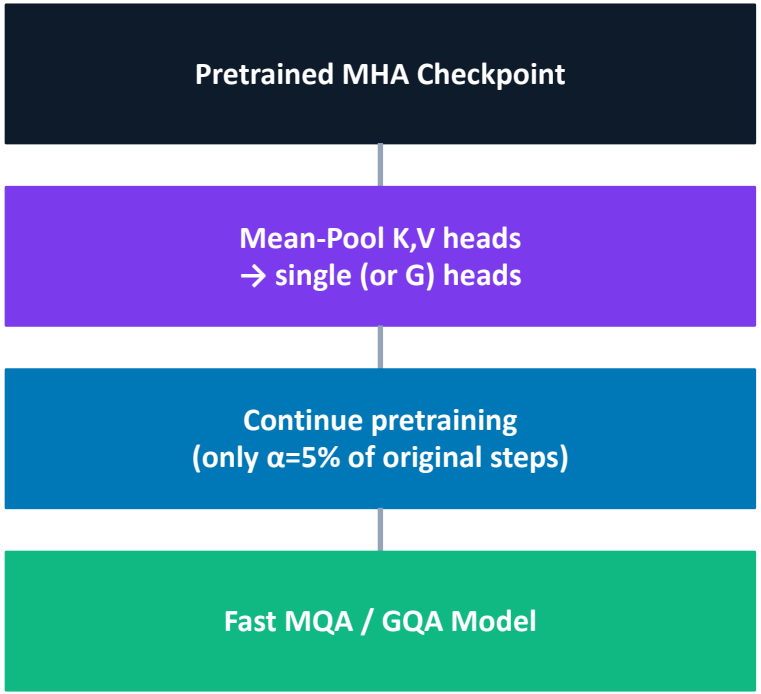
The mechanism: reduce the number of K,V projection matrices — Q heads remain unchanged





Uptraining: Don't Start From Scratch

Conversion Pipeline



Why Mean-Pool? (Fig 4)

Mean: 54.8 ✓best
First: 54.6 (drops info)
Random: 54.4 (forgets all)

Preserving information from all heads matters — mean pooling retains the most signal.

Cost: Just 5% of Pretraining

≈ 600 TPUv3 chip-days vs 12,000+ for full training. You get a new, faster model for almost free.

From MHA to MQA: Mean Pooling, Uptraining, and KV Cache Efficiency

Technical Reference · Attention Mechanism

① PRETRAINING (MHA)

WEIGHT MATRICES

Head 1	Head 2	Head 3	Head 4
W_K^1	W_K^2	W_K^3	W_K^4
W_V^1	W_V^2	W_V^3	W_V^4

PROJECTIONS

$$K^1 = X\ W\_K^1$$
$$K^2 = X\ W\_K^2$$
$$K^3 = X\ W\_K^3$$
$$K^4 = X\ W\_K^4$$

$$V^1 = X\ W\_V^1 \quad \dots \quad V^4 = X\ W\_V^4$$

ATTENTION (HEAD I)

$$A_i = \text{softmax}(Q_i K_i^T / \sqrt{d\_h})\ V_i$$

KV CACHE SHAPE

KV Cache: [t, h , d_h]

Each head has independent K and V projections → large KV cache proportional to h

② CONVERSION (MEAN POOLING)

MEAN POOLING — K PROJECTIONS

W_K^1 W_K^2 W_K^3 W_K^4 → ⊕ ÷ 4 → W_K^{MQA}

MEAN POOLING — V PROJECTIONS

W_V^1 W_V^2 W_V^3 W_V^4 → ⊕ ÷ 4 → W_V^{MQA}

FORMULAE

$$W\_K^{MQA} = 1/h \cdot \Sigma_i\ W\_K^i$$
$$= (W\_K^1 + W\_K^2 + W\_K^3 + W\_K^4) / 4$$
$$W\_V^{MQA} = 1/h \cdot \Sigma_i\ W\_V^i$$
$$= (W\_V^1 + W\_V^2 + W\_V^3 + W\_V^4) / 4$$

RESULT

h K-heads → **1** shared W_K
h V-heads → **1** shared W_V
W_Q per-head: unchanged

Mean pooling combines head-specific weights into one shared projection. One-time operation; no pooling at inference.

③ UPTRAINING

ARCHITECTURE (MQA)

Q^1 Q^2 Q^3 Q^4 → **Single W_K · Single W_V**

GRADIENT UPDATES

$$\downarrow W\_K \leftarrow W\_K - \eta \nabla\_W\_K\ L$$
$$\downarrow W\_V \leftarrow W\_V - \eta \nabla\_W\_V\ L$$
$$\downarrow W\_Q^i \leftarrow W\_Q^i - \eta \nabla\_W\_Q^i\ L$$

TRAINING OBJECTIVE

L = CrossEntropy(y, ŷ)
Same as pretraining loss
a fraction of original compute

EFFECT ON W_K, W_V

Pooled init → fine-tuned
shared weights adapt to
serve all Q heads jointly

Model adapts shared K/V through further training. No pooling occurs here — gradients flow through a single W_K, W_V.

④ INFERENCE (MQA)

SINGLE SHARED PROJECTION

$$K = X\ W\_K$$
$$V = X\ W\_V$$

ATTENTION FAN (ALL Q → SAME K, V)

KV CACHE SHAPE

KV Cache: [t, 1 , d_h]

CACHE REDUCTION FACTOR

MHA: t × h × d_h × 2
MQA: t × 1 × d_h × 2

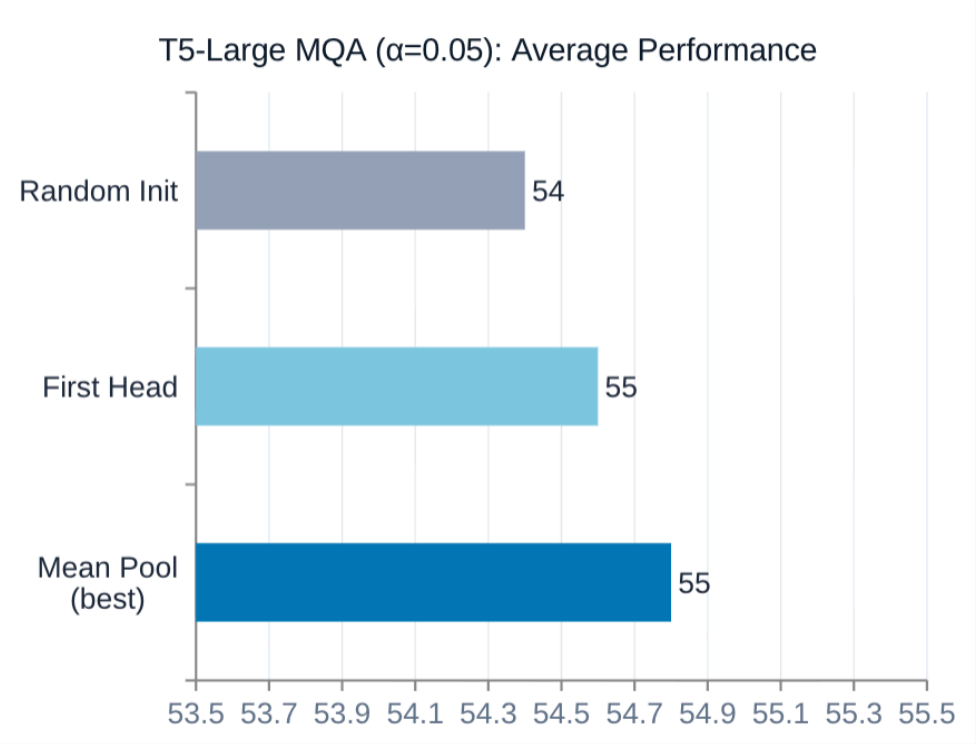
Reduction: **h×** (e.g. 32× for h=32)

Shared K/V reduces memory bandwidth and KV cache size by h× → faster autoregressive decoding.

KEY TAKEAWAY

Mean pooling is a one-time weight transformation applied after pretraining to convert MHA into MQA — it averages all h head-specific K/V projection matrices into a single shared W_K, W_V. **Uptraining** then adapts the model to these shared weights via continued gradient descent. The result: a **h× smaller KV cache** at inference, enabling significantly faster autoregressive decoding with minimal quality loss.

Fig 4: How You Initialize the New Heads Matters



Why the ordering?

More preserved info → better start

Mean: aggregates signal from all original K/V heads

First: keeps one, discards the rest

Random: starts blind — must re-learn everything from 5% data

How KV Caches Are Merged

Each head specializes in a different context type. MQA/GQA merge these into shared KV projections using mean pooling of the original weight matrices.

MHA

4 independent KV caches

KV Cache₁ — Syntax context
'cat' ← subject of 'sat'

KV Cache₂ — Semantic context
'sat' ~ resting/positioned

KV Cache₃ — Coreference
'it' → resolves to 'cat'

KV Cache₄ — Logical
'on mat' → location of action

MQA

All 4 heads → 1 KV Cache (mean pooled)

Q Q Q Q

1 Merged KV Cache

$\text{mean}(K_1, K_2, K_3, K_4)$
$\text{mean}(V_1, V_2, V_3, V_4)$

⚠ Blended context
All specializations averaged → quality drops

GQA (G=2)

2 groups × 2 heads → 2 KV Caches

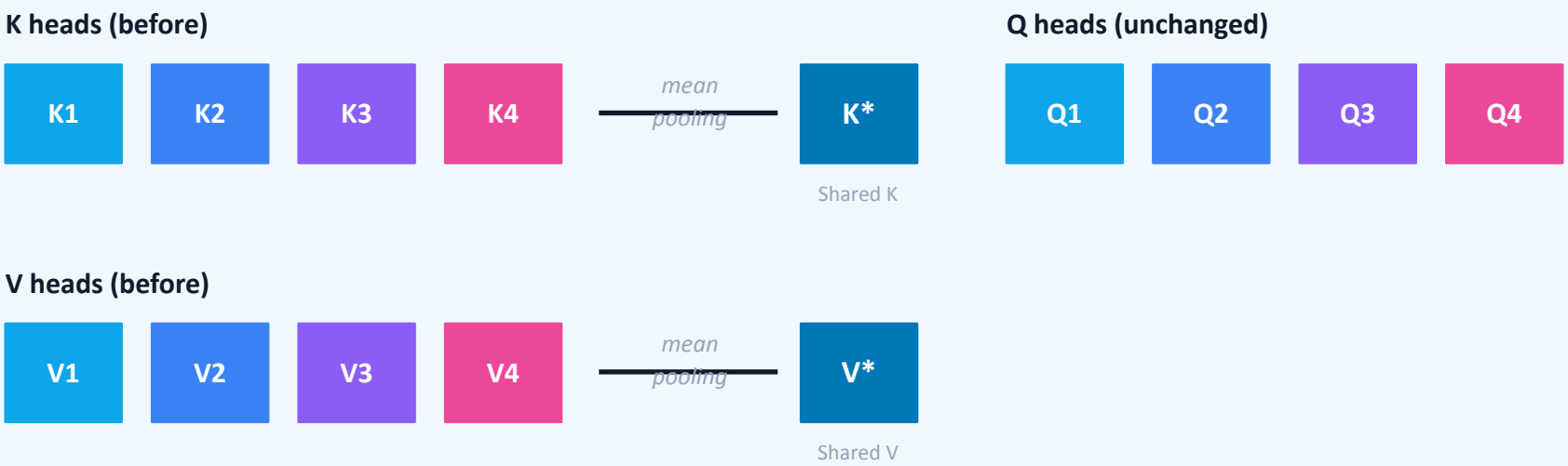
Q_1, Q_2 → **KV-A**
$\text{mean}(K_1, K_2)$
$\text{mean}(V_1, V_2)$
Syntax+Semantic

Q_3, Q_4 → **KV-B**
$\text{mean}(K_3, K_4)$
$\text{mean}(V_3, V_4)$
Coreref+Logic

Mechanism: mean-pool weight matrices
W_K and W_V within each group before uptraining

MHA	GQA-2	MQA	GQA sweet spot: near-MHA quality + near-MQA speed
4 KV Caches	2 KV Caches	1 KV Cache	
100% size	50% size	1.6% size	

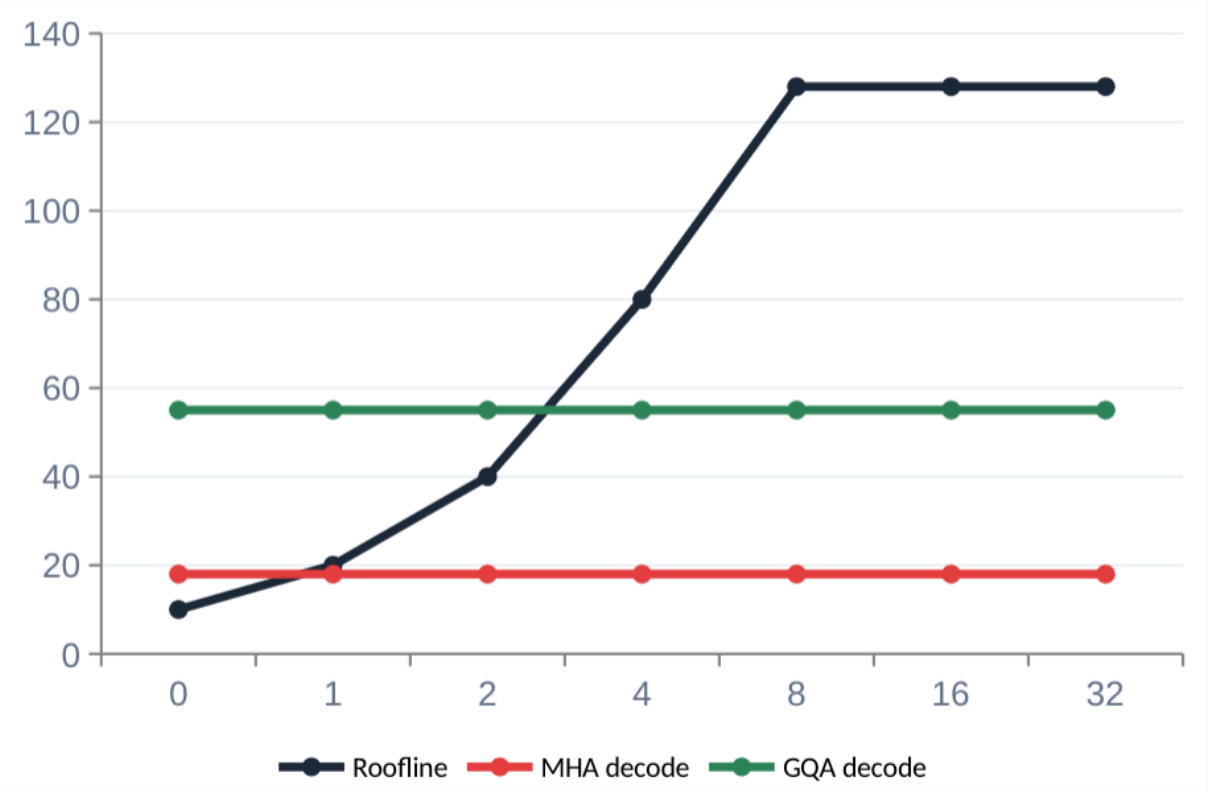
From MHA to MQA: Head Pooling (Fig 1)



Key Insight

Q projections are independent per head (that's fine — queries are computed fresh each step).
But K,V are the cache — we want fewer of them. Mean-pooling merges the "knowledge" from all K heads into one.

The Roofline Model: where decode lives

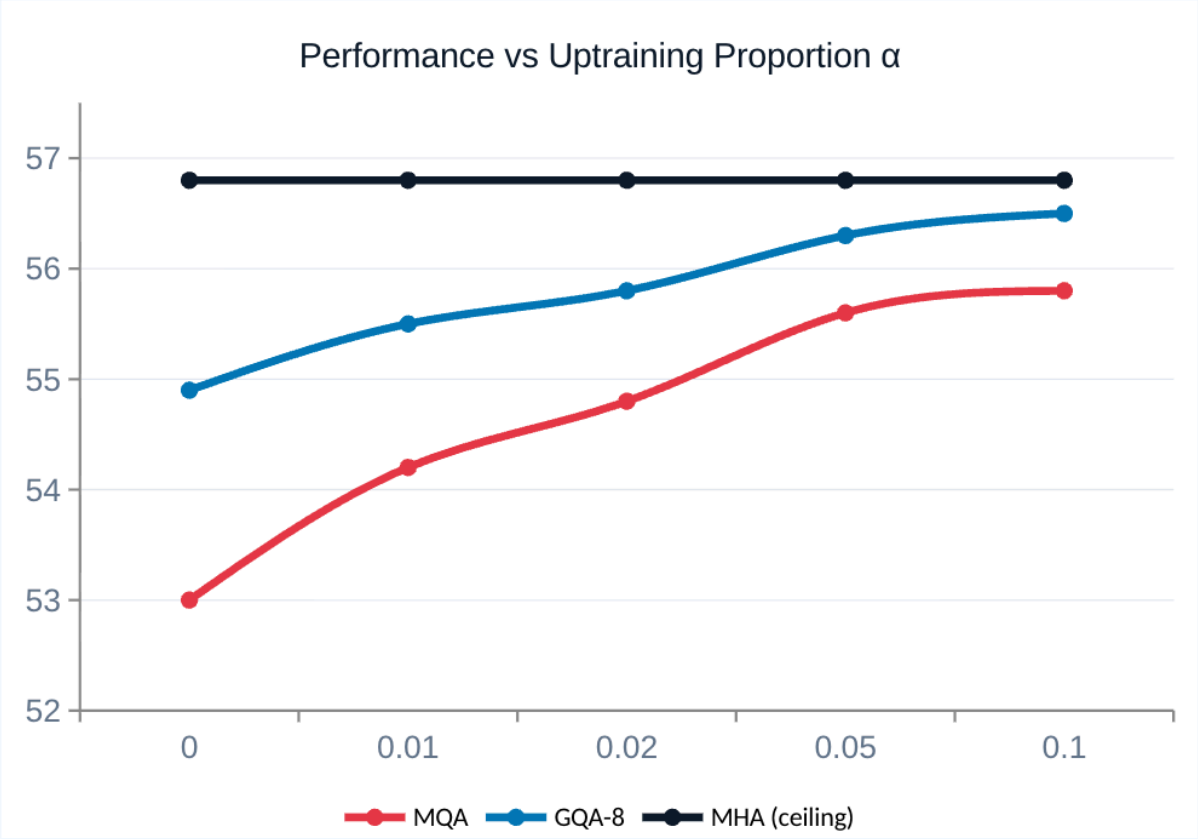


MHA decode: memory-bound
→ spends most time loading KV cache

GQA decode: moves toward compute-bound region

Reducing KV heads → less data to load → higher arithmetic intensity

Fig 5: How Much Uptraining Do You Need?



Q: Why does GQA need less uptraining than MQA? What does this tell us about capacity vs bandwidth?

Takeaways

GQA works even at α=0
(mean pool is already reasonable)

MQA needs uptraining
(pooling to 1 head is lossy)

Both plateau by 5–10%
(diminishing returns past 5%)

Critical threshold: 0.05
Sweet spot: big gains, low cost

Main Results: Quality vs Speed (Fig 3 + Table 1)

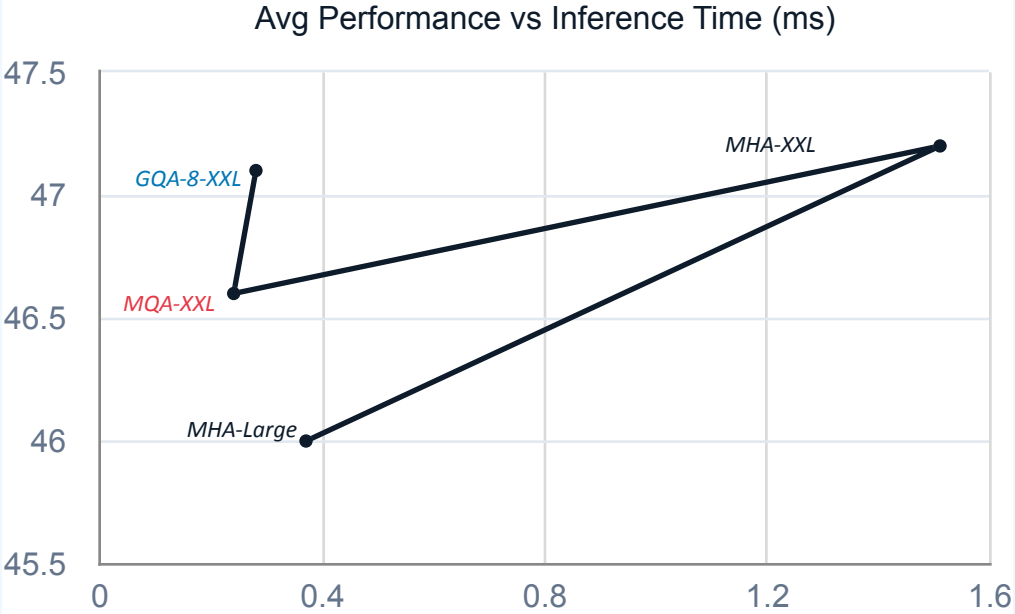


Table 1 summary

Model	Time(s)	Avg
MHA-Large	0.37	46.0
MHA-XXL	1.51	47.2
MQA-XXL	0.24 ✓	46.6
GQA-8-XXL	**0.28 ✓**	**47.1 ✓**

What This Means

GQA-8-XXL hits the sweet spot:

Speed: 6× faster than MHA-XXL, comparable to MQA

Quality: Only 0.1 pts below MHA-XXL on average

Practical win: Serves bigger models at same latency budget

Tradeoff to discuss: We don't have a GQA-8-XXL trained from scratch to compare against!

Tradeoffs

✓ What Improved

- 6× faster decoder inference
- KV-cache size divided by H/G
- Stable training (vs MQA instability)
- Works with existing checkpoints

~ What Stayed the Same

- Q-projection compute (recomputed every step)
- Encoder attention (not the bottleneck)

✗ What Didn't Get Addressed

- No scratch training comparison for GQA
- Only tested on T5 (enc-dec), not decoder-only

? Questions

- You're designing a new 70B model from scratch. Do you use MHA, MQA or GQA? What trade-offs would you consider?
- Why did the paper never compared GQA to a model trained from scratch with GQA?

-> They explicitly cite compute limitations. Training a 11B-parameter T5-XXL from scratch with GQA architecture would require 12,000+ TPUv3 chip-days. The uptraining approach only needed 600. They made a pragmatic choice.

Memory bandwidth vs. capacity: At 70B parameters, the model has roughly 64 attention heads with d_head=128.

With MHA, the KV cache for a single request at 4K context is ~2GB. That means at batch size 8, you're loading 16GB of KV data every single decode step before any computation happens. The GPU is completely memory-starved. MQA solves this by collapsing to 1 KV head — 64× smaller cache — but at 70B the model needs that representational capacity. One KV head simply cannot capture the diversity of attention patterns that 64 query heads are trying to express.

GQA-8 gives you 8 KV heads — an 8× reduction in KV cache size — while preserving enough representational capacity that the quality gap from MHA is negligible. Llama 2 70B specifically chose G=8 for exactly this reason.

The uptraining gap between them suggests that **the quality degradation in MQA is primarily a capacity problem, not a bandwidth problem.**
If it were purely a bandwidth issue, both MQA and GQA would recover at the same rate with uptraining.
But GQA needs much less uptraining, which means it started with more representational capacity intact.
The model doesn't need to learn new attention patterns from scratch — it just needs to fine-tune existing ones.

GQA in the Wild — Why It Matters Beyond the Paper

2019	2022	2023 May	2023 Jul	2024+
Shazeer proposes MQA (1 KV head)	PaLM ships with MQA but closed source	This paper: GQA + uptraining recipe	LLaMA 2 adopts GQA for 34B, 70B models	Mistral, Gemma, Qwen all use GQA by default

The Broader Lesson

You don't need to retrain from scratch. The uptraining recipe lets organizations convert their existing expensive MHA checkpoints.

GQA solved a practical gap: MQA was known to be fast but unstable; many orgs avoided it. GQA gives most of the speed with almost none of the instability.

Today, GQA is the default in virtually every major open-source LLM. This paper is why.

The Instability Problem (Appendix A)

MQA: Training Instability

Observed:
- Frequent loss spikes during pre-training
- Models diverge immediately on long-input fine-tuning
- Required 3-run averages to get stable metrics

Root cause (hypothesis):
Single K,V head must simultaneously serve all Q heads — gradient conflicts when heads want to encode different information in the same projection.

Practical consequence:
Many teams shipped MHA or avoided MQA entirely despite the speed benefits.

GQA: Stability Restored

Observed:
- GQA uptrained models: stable across all tasks
- Single fine-tuning run sufficient
- No divergence even on long inputs (2048 tokens)

Why more stable?
Groups allow K,V heads to specialize. G groups → G sub-problems, each with less conflicting gradient pressure.

Unstated implication:
Stability alone might justify GQA over MQA even if speed gains were identical.

References

Ainslie, J., Lee-Thorp, J., de Jong, M., et al. (2023). GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. arXiv:2305.13245.

Shazeer, N. (2019). Fast Transformer Decoding: One Write-Head is All You Need. arXiv:1911.02150.

Pope, R., Douglas, S., Chowdhery, A., et al. (2022). Efficiently Scaling Transformer Inference. arXiv:2211.05102.

Raffel, C., et al. (2020). Exploring the Limits of Transfer Learning with T5. JMLR 21, 140:1–67.

Touvron, H., et al. (2023). LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971.

Dao, T., et al. (2022). FlashAttention: Fast and Memory-Efficient Exact Attention. arXiv:2205.14135.

Komatsuzaki, A., et al. (2022). Sparse Upcycling: Training Mixture-of-Experts from Dense Checkpoints. arXiv:2212.09596.

Chowdhery, A., et al. (2022). PaLM: Scaling Language Modeling with Pathways. arXiv:2204.02311.